Comparative Analysis of Randomized Optimization Algorithms

Introduction

Randomized optimization algorithms have proved to be a useful technique for optimizing the performance of learning models. By methodically moving to better positions within a search space, randomized optimization algorithms can iteratively optimize learner outputs without the use of gradient methods. In this paper, we will investigate two problem sets while exploring the performance of randomized optimization algorithms such as randomized hill climbing (RHC), simulated annealing (SA), genetic algorithms (GA), and mutual information maximization for input clustering (MIMIC). Through analysis, it will apparent that the usefulness of an optimization algorithm will be dependent on what type of problem it is trying to solve. Each problem set and algorithm will be described in greater detail in subsequent sections below.

Randomized Optimization Algorithm Overview

Randomized Hill Climbing (RHC)

RHC searches for local optima by stepping towards neighboring positions which yield better fitness. Once a peak (or local maxima) is reached, the algorithm randomly selects another position to evaluate as the maximum. If the newly selected position is greater than the current maximum, the position/maximum is updated. If the current optimal value is greater than the new one, the algorithm will continue its random search looking for better outputs. This process will be repeated until no further improvements can be found. The simplicity of this algorithm makes it computationally inexpensive per iteration, however it could be slow to converge in complex spaces.

Simulated Annealing (SA)

The SA algorithm is similar to RHC in that it will evaluate whether a neighboring point or a random point is better than the current optimum, and if it is better fit, it will move to that new point. However, if the randomly selected point value is less than the current optima, the algorithm will calculate its probability to move to the new point based on an acceptance function. This is an exponential function which evaluates the difference in fitness between the current point and the new point. The difference is then divided by a temperature value (T) which will adjust the acceptance criteria for moving to the new point. By fine tuning the temperature value over iterations by using a cooling rate, the algorithm will balance exploration for the new space and exploitation of the current space to find an optimal solution.

Genetic Algorithms (GA)

Genetic Algorithms are a class of algorithms which find an optimum solution based on crossover (or mating) and mutating. The algorithm initially generates a population of random solutions by converting features into bit strings. The algorithm then samples the best solutions from a fitness distribution and generates new solutions by mixing local aspects (a bit sequence) of sampled solutions together two at a time. This process is known as crossover, and the notion here is that the new “breed” of solutions may hold the best aspects of the two previous best solutions. Furthermore, another component of randomness is added to the algorithm through mutation, where a random bit value may be flipped to help convergence (although it can also hinder convergence). Genetic algorithms prove to be useful where the optimal

solution is dependent on the locality of bits within an encoded bit string. If this structure is not apparent, then it the algorithm may fail to converge to a good optimum.

Mutual Information Maximization for Input Clustering (MIMIC)

The MIMIC algorithm finds optimum values based on joint probability density functions of the input features or attribute values. The notion here is that there is some information about any particular feature will affect the remaining features. By modeling them probabilistically relative to the fitness value, the algorithm can build a sense of structure of the optimal solution space through iteration. MIMIC will initially generate a population (or sample size) of random solutions. By comparing their fitness values MIMIC will then keep the best of the solutions (given by a keep value input parameter) and build a new probability distribution to sample from. This process will be repeated until an optimal solution is found, given by a probability distribution of 1. Probability distributions in MIMIC can be represented with dependency trees, where each feature/attribute will be conditionally dependent on its parent. The algorithm also includes a probability constant (M) hyperparameter which is added to each node in the dependency tree to adjust the probability and dependency of a child node on its parent node. A higher M will make child nodes more dependent on the parent node, and a lower M value will have to opposite effect. Disadvantage of this algorithm is that it requires long runtime per iteration, and it can converge to a local minimum in complex spaces where an underlying structure is difficult to find.

Problem Set 1 – Problem Overview (Neural Network (NN) with Randomized Optimization)

A neural network was constructed using a spam classification dataset taken from the UCI data repository. The dataset uses 57 attributes over 4601 instances (or samples) to characterize spam data for a user named George Forman. Each attribute is a measure of the frequencies of a particular word or character in the email. Using this data set, randomized optimization algorithms RHC, SA, and GA will be used to train and find the optimal hypothesis of a neural network. Additionally, the results of each randomized optimization algorithm will be compared to the standard NN backpropagation algorithm trained with cross validation. Layer sizes of the neural network were taken from an assessment done in Assignment 1, where the backpropagation model optimized in accuracy for this dataset.

Problem Set 1 - Methodology

Below are the hyperparameters tested for each of the optimization algorithms implemented with the neural network. Each combination of hyperparameters was tested once. The randomized optimization algorithms were implemented using the ABAGAIL library in Jython.

- Randomized hill climbing hyperparameters tested: None
- Simulated annealing hyperparameters tested:
    - Cooling Exponents values: 0.15, 0.35, 0.55, 0.70, 0.95
- Genetic algorithm hyperparameters tested:
    - Population size: 50
    - Number of crossovers: 10, 20
    - Number of mutations: 10, 20
- Neural network hyperparameters tested:
    - Input Layer size: 57
    - Hidden Layers: 2 layers, 57 nodes each

- Output Layer size: 1

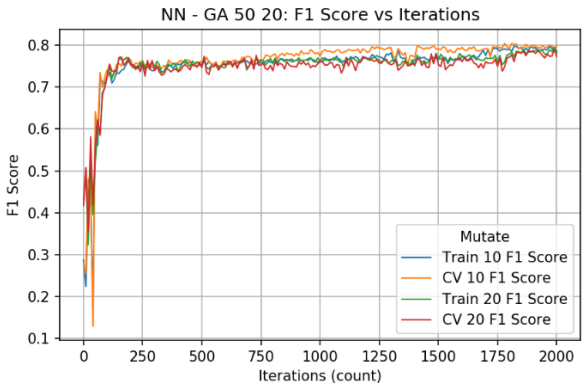
Problem Set 1 - Results

The hyperparameters that yielded the highest F1 score accuracy for each algorithm are shown in Table 1 below.

Table 1. Best Tested Hyperparameters for Problem Set 1 Algorithms

Algorithm	**Hyperparameter Values**
RH	N/A
SA – Cooling Exponent	0.35
GA – (Population, # of Crossovers, # of Mutations)	(100, 20, 20)
Backpropagation – (Hidden Layer 1, Hidden Layer 2)	(57 ,57)

Table 1 above implies that simulating annealing algorithm performed best with a cooling exponent of 0.35 and the GA performed best with 20 crossovers and mutation during each iteration. However, it can be seen in Figure 1 below that GA performed well for each of the input hyperparameters tested. Figure 2 shows comparable results in that SA cooling exponents 0.15, 0.35, 0.55, and 0.70 all yielded similar results after 2000 iterations. The hyperparameters both of these optimization algorithms proved to have a little effect on accuracy.



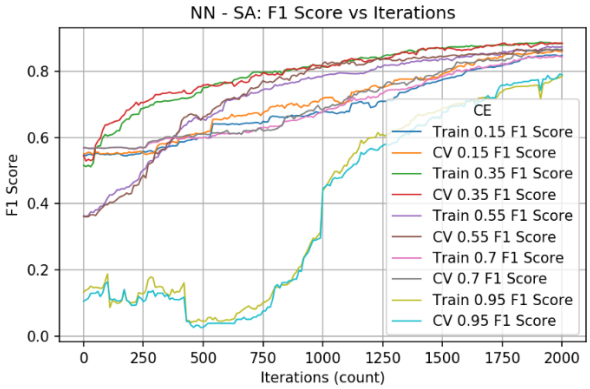


Figure 1. GA Tested Hyperparameter Performance **Figure 2. SA Tested Hyperparameter Performance**

The F1 scores for the top performing optimization algorithm hyperparameters were calculated over N = 5000 iterations. The results in Figure 3 below highlight the scores of each algorithm until they all converged (2000 iterations).

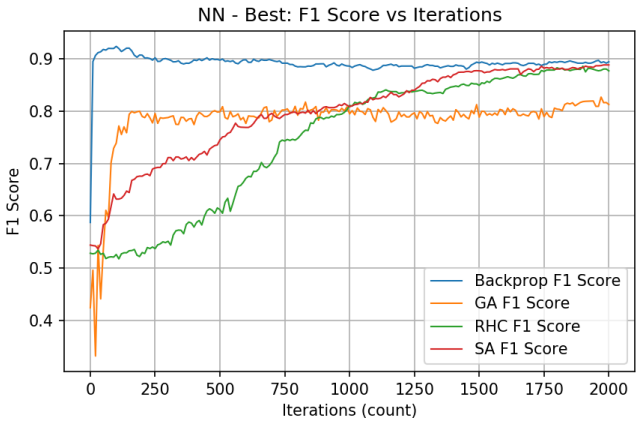


Figure 3. F1 Scores vs Iterations

Figure 3 indicates that backpropagation had the best overall performance in terms of accuracy. Backpropagation was the fastest to converge over iterations and consistently had the highest F1 score. This algorithm only converged around the highest score over N iterations, meaning that the algorithm did not get stuck at any local optimum that any of the *other* optimization algorithms did. It is suspected that the rebalancing of weights across all attributes (and by association every node in the network) relative to their magnitude of error can force a particular attribute stuck at a local optimum to move out of such a space.

In terms of F1 score accuracy, SA and then RHC performed the second and third best respectively. The results of these two algorithms were similar, while SA consistently outperformed RHC until about 1000 iterations, both algorithms didn't finish converging until around 2000 iterations. Furthermore, the accuracies of both algorithms would slowly progress across iterations since they would continually reach local optima. This is expected as both are instance-based algorithms and they must begin to converge by chance as the number of iterations increase.

Genetic algorithm performed the worst in terms of accuracy over the number of iterations. The algorithm converged after 490 iterations with an F1 score of 0.7978. Due to this lower convergence value, it can be suspected that crossover and mutation of the features may have had destructive effects, where the algorithm converged towards a local optima and additional crossovers and mutations yielded no new useful information. This is to be expected as the solution space for this algorithm is complex and doesn't rely on locality. It will be difficult for a GA algorithm to produce the optimal output through crossover and mutation. It may be possible to increase the score of the GA by introducing more randomness into the algorithm to land on better optima. This can be done by increasing the initial population size, number of crossovers, and the number of mutations.

To weigh the benefit of each algorithm with respect to time and error, a comparison between each algorithm's compute time, convergence time, and accuracy metrics is shown in Table 2 below.

Table 2. Max iterations Execution Times, Convergence Times, and Convergence Accuracy

Algorithm	**Time after 5000 Iterations (s)**	**Approximate Convergence Time (s)**	**Convergence F1 Score (Test Set)**
Backpropagation	716.72	34.58	0.8969

SA	230.18	91.57	0.8877
RHC	222.83	89.34	0.8764
GA	6853.35	641.80	0.7978

Table 2 suggests that the Backpropagation algorithm is the performed best for convergence time and accuracy. SA and RHC algorithms performed next best and would result in accurate solutions if given enough run time. The GA converged the quickly over iterations, but the iterations where computationally expensive, resulting in the longest convergence time; when coupled with its low F1 score, GA proved to be the least useful optimization algorithm for this problem.

Problem Set 2 – Problem Set Overview (Optimization Algorithms against Optimization Problems)

Flip Flop, Continuous Peaks, and Traveling Sales Person (TSA) optimization problems were evaluated using the RHC, SA, GA, and MIMIC algorithms. Each optimization problem used a distinct fitness function to evaluate how well each of the optimization algorithms performed over iterations. Fitness times, fitness scores, and problem complexity were considered when analyzing optimization algorithms. Each of the optimization problems and the methodology used to test the optimization algorithms are described in the subsequent sections below.

Problem Set 2 – Methodology

Below are the hyperparameters tested for each optimization algorithm used to evaluate the optimization problems. Each combination of hyperparameters was tested five times and the results were averaged. The randomized optimization algorithms were implemented using the ABAGAIL library in Jython.

- Randomized Hill Climbing inputs tested: None
- Simulated Annealing inputs tested:
    - Cooling Exponents values: 0.15, 0.35, 0.55, 0.75, 0.95
- Genetic Algorithm Inputs tested:
    - Population size: 100
    - Number of crossovers: 50, 30, 10
    - Number of mutations: 50, 30, 10
- MIMIC Algorithm Inputs Tested:
    - Sample size: 100
    - Keep size: 50
    - M: 0.1, 0.3, 0.5, 0.7, 0.9

Problem Set 2 – Optimization Problem Descriptions and Results

The hyperparameters for each algorithm which performed best in terms of accuracy are shown in Table 3 below.

Table 3. Best Algorithm Inputs for Optimization Problems

	Flip Flop	**Continuous Peaks**	**TSP**
RH	N/A	N/A	N/A
SA	0.55	0.35	0.35
GA	(100, 50, 50)	(100, 50, 50)	(100, 30, 10)

MIMIC – (# Samples, Keep Size, M)	(100, 50, 0.9)	(100, 50, 0.7)	(100, 50, 0.1)

Although Table 3 highlights the best hyperparameters for RHC, SA, and GA, it is worth noting that each combination of hyperparameters for SA and GA yielded similar results in relation to the optimization problem being solved. However, in the case of MIMIC, the probability constant (M) produced different results depending on the complexity of the problem trying to be solved. The highest value of M yielded the best results for the Flip Flop problem (least complex), and the lowest value of M yielded the best results for the TSP problem (most complex). Figures 4, 5, and 6 below show the best value for M in each optimization problem tested. The complexity of each optimization problem will be discussed in more detail subsequently.

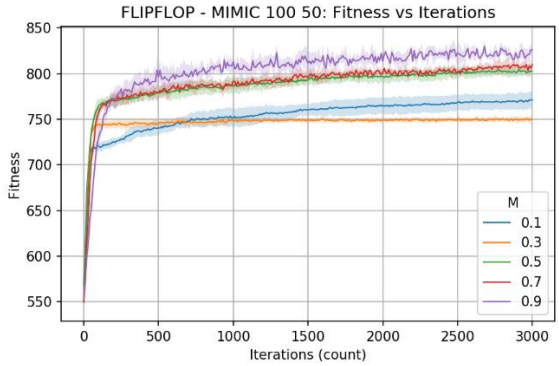


Figure 4. Flip Flop MIMIC Tested Hyperparameters

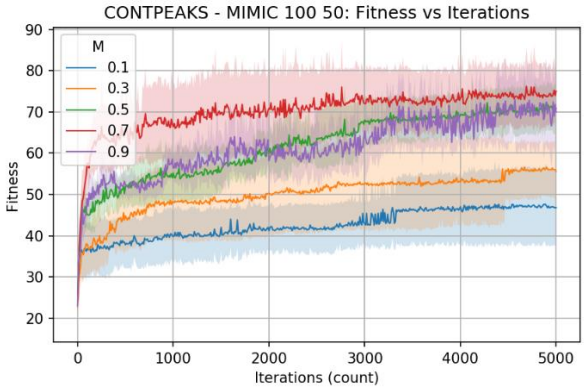


Figure 5. Continuous Peaks MIMIC Hyperparameters

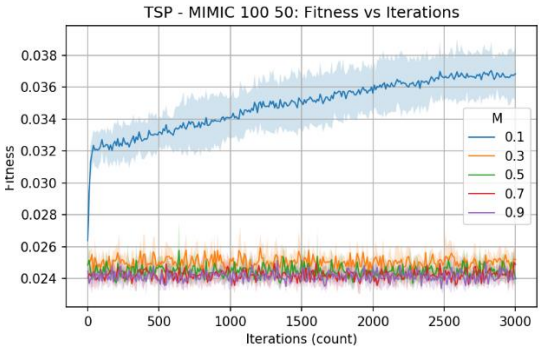


Figure 6. TSP MIMIC Tested Hyperparameters

Flip Flop Optimization Problem

The flip flop optimization problem takes a bit sting length (N) as an input and it evaluated against a fitness function such that a higher score will be achieved if each bit is different from its neighbors. For a perfect fitness score, the bit string will have to alternate between 0's and 1's for each element in the bit string. Therefore, this optimization problem has exactly two global maxima and many local maxima. The simple fitness function used in Flip Flop results in the least complicated solution space of the three algorithms being analyzed. For this assignment, a bit string length of 1000 was used as an input (with max fitness

score of 1000) over 3000 iterations for each optimization algorithm being evaluated. The performance of each algorithm can be seen in Figure 7 below.

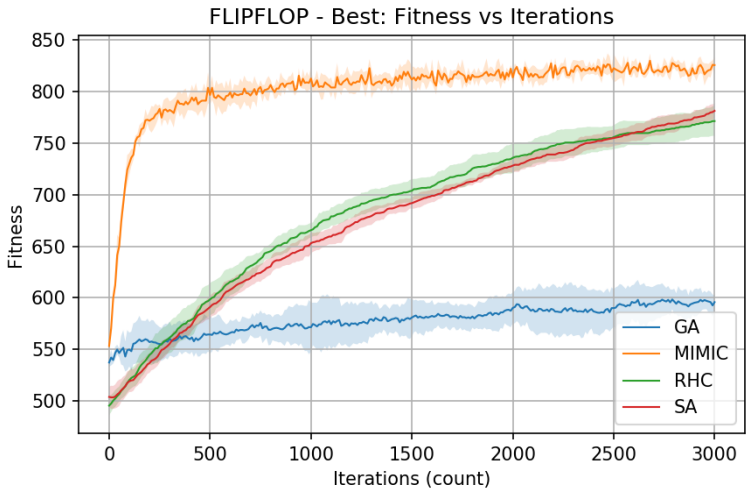


Figure 7. Fitness vs Iterations for Flip Flop Optimization Problem

Figure 7 above illustrates that MIMIC performed best after 3000 iterations. MIMIC was able to converge to a local optimum much faster than the other three algorithms (of which SA and RHC did not converge). The simple underlying structure for the Flip Flop problem proved best for MIMIC. The algorithm was able to use joint probability distributions to emphasize the important aspects of the underlying repeated structure of the solution. Hence, a higher M hyperparameter resulted in the best performance for this algorithm since child nodes (or the current bit value) will be more dependent on parent nodes (or the previous bit value).

Simulated Annealing and Randomized Hill Climbing performed similarly to each other in that they would solve for many local maxima and slowly improve their fitness scores. In the case above, RHC and SA algorithms did not converge, but if given enough iterations both algorithms can land on the optimal solution by chance.

The genetic algorithm converged with the lowest fitness value, indicating that mating and mutating between optimal bit streams yielded insignificant information. This is expected as the locality of different spaces in the bit stream would not result in a high fitness score. To try to improve the performance of GA, introducing more randomness, given by a larger initial population size, more crossovers, and more mutations, may increase the probability of outputting a better solution.

The wall clock times of each algorithm can be seen in Table 4 below.

Table 4. Flip Flop Algorithm Performance Time for Max Iterations (3000)

Algorithm	**Average Time for Max Iteration (s)**
Simulated Annealing	0.0222
Randomized Hill Climbing	0.0335
Genetic Algorithms	0.8364
MIMIC	1921.3265

Although MIMIC performed best when fitness was dependent on iteration count, it took orders of magnitude longer to perform than the other three optimization algorithms. In the context of solving the problem the fastest, SA and RHC will likely have return much higher fitness scores if given the same amount of time to run as MIMIC.

Continuous Peaks Optimization Problem

The continuous peaks optimization problem takes a bit string length (N) and a threshold value (T) as inputs. To get the maximum fitness score for this problem, the number of consecutive, equivalent bit values starting from the first index must stop at index T+1, and remainder of the bit sequence must have the opposite bit value. The problem gives a reward value equal to N depending on if consecutive 0-valued bits and 1-valued bits the length of T can be found in the solution. For evaluation and to further increase the complexity of the solution space, T=29 and N=100 hyperparameters were chosen for evaluation. The highest fitness score given these inputs would be 170. The performance of the optimization algorithms over 5000 iterations can be seen in Figure 8 below.

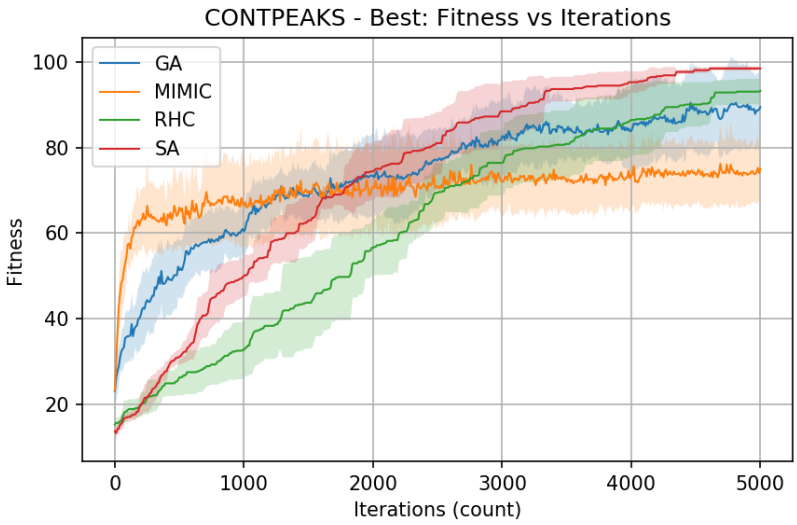


Figure 8. Fitness vs Iterations for Continuous Peak Optimization Problem

It can be seen from Figure 8 above that each algorithm scored low in relation to the maximum possible fitness score of 170. This is expected since the T value increased the solution space complexity. However, when comparing each score relative to one another, SA performed best. The nature of the continuous peaks problem presents many locally optima solutions of different magnitudes. Simulated Annealing will work best for this problem since the algorithm can search for values similar to its current local optima and step towards a local or global optima. As the local optima values improve, temperature will decrease, narrowing the searchable locations of new points to be similar to the new local optima.

RHC performed second best at 5000 iterations since it is by natures similar to SA, and then GA and MIMIC performed 3rd and 4th best respectively. It is expected for GA and MIMIC to perform the worst since an underlying structure and a relation to locality within the optimal bit sequence is difficult to find given its high threshold value of 29. The structure requires balancing the tradeoff between the maximizing number of consecutive equal-valued bits for both bit values (0 and 1), this is particularly hard to represent with dependency trees used in MIMIC even with a low M value.

Timing performance for this optimization problem was also measured for each algorithm. The results can be found in Table 5 below.

Table 5. Continuous Peaks Algorithm Performance Time for Max Iterations (5000)

Algorithm	**Time for Max Iterations (s)**
Simulated Annealing	0.0283
Randomized Hill Climbing	0.0276
Genetic Algorithms	0.3483
MIMIC	22.8416

Table 6 above highlights that SA and RHC both required the least time to yield the optimum solution after 5000 iterations, proving that SA and RHC are the best algorithms to solve this problem when considering time, iterations, and accuracy. The genetic algorithm performed next best but using this algorithm to solve the problem will only result in higher completion time and lower accuracy. The same can be said for MIMIC with had the highest completion time and the lowest accuracy.

Traveling Sales Person (TSP) Problem

TSP is an NP-hard problem frequently used to evaluate the efficiency of algorithms due to its highly complex solution space. The problem is structured like this, “given a list of cities and a starting place, the goal is to find the shortest path to visit every city and come back to the starting location”. The complexity of the algorithm is driven by the dynamic solution space where adjusting one connection in a given path can have detrimental effects for the remainder of the path. Additionally, the solution spaces consist of many local optima that may be close in fitness value to each other but have significantly different structure. The performance of each algorithm in a TSP problem with 100 points (or cities) over 3000 iterations can be seen in Figure 9 below.

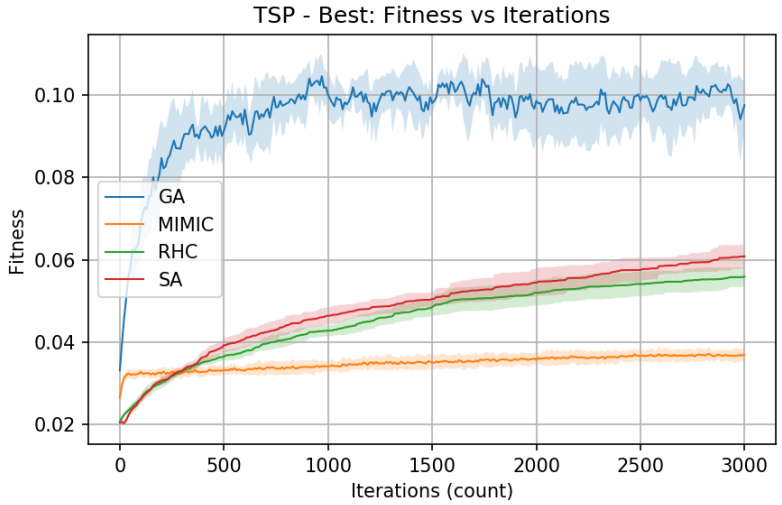


Figure 9. Fitness vs Iterations for TSP Optimization Problem

Figure 9 suggests that genetic algorithms outperformed all other algorithms by a substantial margin. By organizing information by distance and using crossover and mutation amongst similar chromosomes, the GA algorithm was able to fit successful aspects of each chromosome (or city cluster when alluding to

the TSP analogy) to organize low distance points close to one another and yield a high fitness score. In the case of SA and RHC, the algorithms were effectively trying to guess the shortest route by iterating through the solution space. Given that the solution space has many local optima and only 1 global optima, we can see that these algorithms will struggle to quickly converge to a solution. MIMIC performed the worst due to the nature of the TSP problem. The problem lacks the necessary structure for the MIMIC algorithm to work. Changing one connection from your current optima, can drastically change the entire solution along with its fitness value. This cannot be represented with dependency trees even when the nodes of the trees are loosely dependent on one another with a probability constant (M) of 0.1.

Table 6. TSP Algorithm Performance Time for Max Iterations (3000)

Algorithm	**Time for Max Iterations (s)**
Simulated Annealing	0.0161
Randomized Hill Climbing	0.0150
Genetic Algorithms	0.4993
MIMIC	476.9944

Table 6 above shows that GA took the third longest to complete 3000 iterations. However, given its quick convergence at ~800 iterations or 0.1477 seconds, GA are best to solve this problem with a small compromise in speed for a significant improvement in accuracy. The remaining solutions did not perform well relatively to GA, with MIMIC performing the worst with the longest run time and the lowest accuracy.

References

Hopkins, M., Reeber, E., Foreman, G., & Suermondt, J. (1999, January 07). Retrieved February 2, 2019, from https://archive.ics.uci.edu/ml/datasets/spambase

Isbell, C. L., Jr., De Bonet, J. S., & Viola, P. (1997). MIMIC: Finding Optima by Estimating Probability Densities. Retrieved February 29, 2019, from https://papers.nips.cc/paper/1328-mimic-finding-optima-by-estimating-probability-densities.pdf

Isbell, C. L., Jr., Randomized Local Search as Successive Estimation of Probability Densities. Retrieved February 29, 2019, from https://www.cc.gatech.edu/~isbell/tutorials/mimic-tutorial.pdf

Pushkar. (2018, March 08). Pushkar/ABAGAIL. Retrieved February 27, 2019, from https://github.com/pushkar/ABAGAIL